Hacettepe University

Department of Artificial Intelligence

Doctor’ s Aid Assignment Report

Course Name:

BBM103 – Introduction to Programming Lab

Assignment 2

Report Prepared by:

Özge Bülbül

2220765008

November 24, 2022

Table of Contents

1 Introduction

Clinical decision support systems (CDSS) are computer-based programs that analyze data within EHRs to provide prompts and reminders to assist health care providers in implementing evidence-based clinical guidelines at the point of care. CDSS encompasses a variety of tools to enhance decision-making in the clinical workflow. These tools include automatic alerts and reminders to care providers and patients, clinical guidelines, condition-specific order sets, focused patient data reports and summaries, documentation templates, diagnostic support, and contextually relevant reference information, among other tools. Therefore, artificial intelligence can be used in the field of medicine this way to make the physicians' and nurses' jobs quite easier. In this assignment, it is focused to create a probability-based decision system called Doctor's Aid for clinicians.

2 Analysis

Approximately 19.3 million patients were diagnosed with cancer and 10 million patients died due to cancer in 2020. For instance, with breast cancer, prevention, early detection and treatment are possible via mammographic screening. Nevertheless, there are still some obstacles faced during this process such as overdiagnosis and overtreatment. The Doctor's Aid system is created to prevent these obstacles.

The Doctor's Aid system provides the clinician with the requested patients' name, diagnosis accuracy, disease name, disease incidence, suggested treatment's name and treatment's risk. The system also calculates the patients' probability of actually having the assumed cancer type and by using the patients' disease probability and the suggested treatment's risk, it makes recommendations on whether the treatment should be taken by the patients' or not.

3 Design

3.1 Reading and Writing the Data

In the Doctor’s Aid system, the data is supposed to be taken from and written to a .txt file. This is applied by importing os module and using the following part of the code.

```
current_dir_path = os.getcwd()
reading_file_name = "doctors_aid_inputs.txt"
reading_file_path = os.path.join(current_dir_path, reading_file_name)
writing_file_name = "doctors_aid_outputs.txt”
writing_file_path = os.path.join(current_dir_path, writing_file_name)
```

This way, the data is read from the “doctors_aid_inputs.txt” file and written to “doctors_aid_outputs.txt” file. In addition to this, a save_output function is defined to write the data to the .txt file and a read_input function is defined to read the data from the .txt file.

```
def save_output(output):
    with open(writing_file_path, 'a') as f:
        f.write(output)
```

```
def read_input():
```

3.2 Creating the New Patients

The Doctor’s Aid system is expected to make a list of every patient that is created by the clinicians. Adding these patients is done by commands such as “create Deniz, 0.9999, Lung Cancer, 40/100000, Radiotherapy, 0.50”. This example means that there is a patient called Deniz who has been diagnosed with lung cancer and this diagnosis method is 99.9% accurate. This cancer type is seen in 40 out of 100000 people in Turkey. Radiotherapy is the suggested treatment and this treatment has a risk of 50%. The “patient_list” list and “patient_create” function were created by me to solve this problem.

```
patient_list = []
```

```
def patient_create(x):
    row = x.split()
    for i in range(len(row)):
        row[i] = row[i].strip(',')
    for patient in patient_list:
        if row[1] in patient:
            save_output("Patient {} cannot be recorded due to
                         duplication.\n".format(row[1]))
            return
    row[3:5] = [' '.join(row[3:5])]
    for i in range(len(row) - 1):
        if row[i] == 'Targeted':
            row[i:i + 2] = [' '.join(row[i:i + 2])]
    row[2] = "{:.2%}".format(float(row[2]))
    row[6] = "{:.0%}".format(float(row[6]))
    patient_list.append(row[1:])
    save_output("Patient {} is recorded.\n".format(row[1]))
```

In the code segment above, first, the patient_list is created. Then, the patient_create function is defined. Inside this function, firstly, elements of the list are splitted into words. Secondly, the commas are removed and then, it is checked if the patient we want to add is already in the list. If so, by using the save_output function that was created previously, "Patient x cannot be recorded due to duplication." is written in the output file. After that, the two words that make the disease name are combined and then if the suggested treatment name is "Targeted Therapy" we combine those two words, too because we want those two words to be just one element that represents a treatment name. Coming to the end of the function, we transform the numbers given to us as diagnosis accuracy and treatment risk to percentages. Then we add the patient's data to our "patient_list" list and write "Patient x is recorded." to our output file.

3.3 Removing the Patients

Here, we need to delete the existing patient's data from the system when we see a "remove" command, for instance, "remove Deniz" command would mean that the clinician wants to delete the data of patient Deniz from the system. If patient Deniz is already not in the list, the system is required to say "Patient Deniz cannot be removed due to absence.".

```
def patient_remove(y):
    row = y.split()
    counter = 0
```

Firstly, in the code segment above, "patient_remove" function is defined and then the elements of the list are separated into words. Secondly, a counter's value is equalized to zero. This counter will later be used in this function for the incidence that the patient that is being tried to remove is already not in the list.

```
for patient in patient_list:
    if row[1] in patient:
        save_output("Patient {} is removed.\n".format(row[1]))
        patient_list.remove(patient)
        break
    else:
        counter = counter + 1
    if counter == len(patient_list):
        save_output("Patient {} cannot be removed due to
                    absence.\n".format(row[1]))
```

In the code segment above, it is first checked if the patient is in the list and if so, “Patient x is removed” is written to the output file and then the patient’s data is deleted from the list. Every time the system checks if the patient name is in the list, if not, the counter goes one up. At the end, if the counter’s value equals to the length of the list, which means that all the elements of the list were checked and none of them were equal to the patient that is being deleted, therefore; this patient is already not in the list. Then, “Patient x cannot be removed due to absence.” is written to the output file.

3.4 Listing All Patients with Their Information

The fifth function we needed to create was the listing function. When called, this function is required to make a list of all the data of each and every patient in the “patient_list”.

```
def make_list():
    save_output('Patient '"Diagnosis\t""Disease \t\t""Disease
                 \t""Treatment\t\t""Treatment\n""Name\t" "Accuracy\t""Name
                 \t\t\t""Incidence\t""Name\t\t\t""Risk\n"
                "----------------------------------------------------------
                 ---------------\n")
```

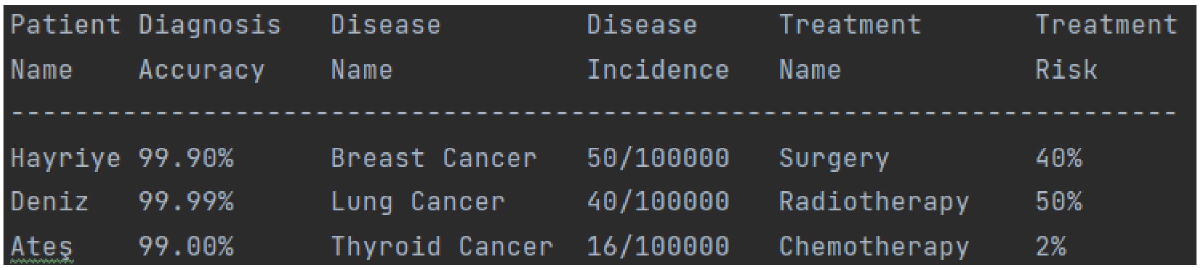
Here, the make_list function is defined and the top part of the table is written so that it looks like the part given below.

```
Patient Diagnosis   Disease          Disease     Treatment        Treatment
Name    Accuracy    Name             Incidence   Name             Risk
-------------------------------------------------------------------------
```

The whitespaces are quite vital during this process and because of that, there are some obstacles we come face to face with. The patient “Su” for example, has a very short name, therefore; requires two tabs between the patient name and diagnosis accuracy and that correction code segment is given below.

```
elif patient[0] == 'Su':
    save_output("{}\t\t{}\t\t{}\t{}\t{}\t{}\n".
                format(patient[0], patient[1], patient[2], patient[3],
                patient[4], patient[5]))
```

Some other obstacles besides Su being too short are Targeted Therapy being too long, Lung Cancer being too short and Surgery being too short. These obstacles are solved the same way. The figure below is an example table that consists of three patients.



```
Patient Diagnosis   Disease         Disease     Treatment       Treatment
Name    Accuracy    Name            Incidence   Name            Risk
-------------------------------------------------------------------------
Hayriye 99.90%      Breast Cancer   50/100000   Surgery         40%
Deniz   99.99%      Lung Cancer     40/100000   Radiotherapy    50%
Ateş    99.00%      Thyroid Cancer  16/100000   Chemotherapy    2%
```

3.5 Patient's Probability of Having the Disease

The "probability" command will be used to call the probability function. Probability function is used to first calculate the patient's actual probability of having the so-called disease and then write this calculated probability to our output file. If we were to get a "probability Deniz" command, the Doctor's Aid system would calculate that Deniz has 80% probability of having the disease and write "Patient Deniz has a probability of 80.00% of having Lung Cancer.".

```
def probability(z):
    lines = z.split()
```

The function is defined and elements of the list are separated into words.

```
for pat in range(len(patient_list)):
    if patient_list[pat][0] == lines[1]:
        disease = patient_list[pat][2]
        accuracy = patient_list[pat][1]
        incidence = patient_list[pat][3]
        accuracy = round(float(accuracy.strip('%')), 4)
        a = round(1 - (accuracy / 100), 4)
        b = incidence.split('/')
        b = float(b[0]) / float(b[1])
        sum = a + float(b)
        prob = float(b) / sum
```

Here, the probability is calculated. The calculation done here is like so: Diagnosis accuracy is a percentage and we transform it into a float and subtract it from 1, disease incidence is a string and we transform it into float by dividing it, then we divide the disease incidence float by the sum of these to floats. This gives us the probability.

```
        global reco
        if reco == 1:
            break
        else:
            save_output("Patient {} has a probability of{} of having
                          {}.\n".format(lines[1], d, disease))
            break
else:
    save_output("Probability for {} cannot be calculated due to
                 absence.\n".format(lines[1]))
```

In the code above, a variable called “reco” is seen. This variable is set to 0 outside the function and is created for the recommendation function that will be explained in subsection 3.6. After that, “Patient x has a probability y of having z.” is written if the patient is in the list. If not, “Probability for x cannot be calculated due to absence." is written to the output file.

3.6 Recommendation for a Particular Treatment

The “recommendation” command will be used to call the recommendation function. The recommendation function compares the patient’s probability of having the disease to the treatment risk and then suggests if the treatment should be applied or not. Patient Deniz has 80% probability of having lung cancer and the suggested treatment, radiotherapy, has 65% risk. In this scenario, the system suggests Deniz to have the treatment and writes “System suggests Deniz to have the treatment.”. Patient Ateş has 1.57% probability of having thyroid cancer and the suggested treatment, chemotherapy, has 2% risk. Therefore, in this case, it writes “System suggests Ateş NOT to have the treatment.” since the risk is higher than the probability.

```
def recommendation(r):
    row = r.split()
```

First the function is defined and elements of the list are separated into words.

```
global probability_number
probability_number = prob.strip('%')
```

The “probability_number” variable is just the “prob” variable without the “%” sign.

```
counter = 0
for pat in range(len(patient_list)):
    if patient_list[pat][0] == row[1]:
        global risk
        risk = (patient_list[pat][5]).strip('%')
        global reco
        reco = 1
        probability(r)
        if float(probability_number) > float(risk):
            save_output("System suggests {} to have the treatment.\n"
                         .format(row[1]))
            reco = 0
```

Above, we first assign our counter to zero, which we will use to see if the patient is in the list or not like we did in the patient_remove function. Then the reco variable is set it to 1. This way, when the probability function is called, the probability won’t be written to the output file. If the probability is higher than the risk, “System suggests x to have the treatment.” Is written. Then reco is equalized to zero to neutralize the code.

```
else:
    save_output("System suggests {} NOT to have the
                 treatment.\n".format(row[1]))
    reco = 0
```

If probability is not higher than the risk, as seen above, “System suggests x NOT to have the treatment.” is written to the output file and again reco is equalized to zero.

```
    else:
        counter = counter + 1
if counter == len(patient_list):
    save_output("Recommendation for {} cannot be calculated due to
                 absence.\n".format(row[1]))
```

Each time the code checks if the patient is in the list, counter goes up by one and if the counter’s value is equal to the length of the list, this means that the patient doesn’t exist in our system and “Recommendation for x cannot be calculated due to absence.” Is written to the output file.

4 Programmer’s Catalogue

In this section, the whole code is provided and then time spent on analyzing, designing, implementing, testing and reporting and the reusability of the code are explained.

4.1 The Code

# Student Name: Özge Bülbül

# Student ID: 2220765008

# OS module is imported to get the current directories for reading and writing the files.

import os

current_dir_path = os.getcwd()

reading_file_name = "doctors_aid_inputs.txt" # The name of the file to read the inputs from.

reading_file_path = os.path.join(current_dir_path, reading_file_name)

writing_file_name = "doctors_aid_outputs.txt" # The name of the file to write the outputs to.

writing_file_path = os.path.join(current_dir_path, writing_file_name)

patient_list = [] # The list that holds all the data of each patient.

```
# This is the 'saving to the output file' function. When called, it writes the output variable to the output file.

def save_output(output):

    with open(writing_file_path, 'a') as f:

        f.write(output)



# This is the 'create a new patient' function. When called, it adds the new patient data to the patient list.

def patient_create(x):

    row = x.split()  # Splits the row into a set of words.

    for i in range(len(row)):

        row[i] = row[i].strip(',')  # Removes the commas.

    for patient in patient_list:  # If the patient's name is already in the patient list, it writes "Patient x cannot be recorded due to duplication.".

        if row[1] in patient:

            save_output("Patient {} cannot be recorded due to duplication.\n".format(row[1]))

            return  # Ends the function and doesn't add the patient ,who is already in the list, to the list again.

    row[3:5] = [' '.join(row[3:5])]  # Unites the two words of the disease name.

    for i in range(len(row) - 1):  # Unites the treatment name words 'Targeted' and 'Therapy' to fix a problem.

        if row[i] == 'Targeted':

            row[i:i + 2] = [' '.join(row[i:i + 2])]

    row[2] = "{:.2%}".format(float(row[2]))  # Makes the diagnosis accuracy a percentage.

    row[6] = "{:.0%}".format(float(row[6]))  # Makes the treatment risk a percentage.

    patient_list.append(row[1:])  # Adds the patient's data to the patient list.

    save_output("Patient {} is recorded.\n".format(row[1]))  # Writes "Patient x is recorded".



# This is the 'remove a patient' function. When called, it removes the existing patient data from the patient list.

def patient_remove(y):

    row = y.split()  # Splits the row into a set of words.
    counter = 0
    for i in range(len(row)):
        row[i] = row[i].strip(',')  # Removes the commas.
    for patient in patient_list:
        if row[1] in patient:  # If the patient is in the list, it writes "Patient x is removed." and
then removes the patient.
            save_output("Patient {} is removed.\n".format(row[1]))
            patient_list.remove(patient)
            break
        else:  # If the patient is not in the list, counter's value would be equal to the length of
patient_list. Which means the patient is not in the list.
            counter = counter + 1
        if counter == len(patient_list):
            save_output("Patient {} cannot be removed due to absence.\n".format(row[1]))



reco = 0
probability_number = None



# This is the 'probability' function. When called, it calculates patient's disease probability and
writes it.
def probability(z):
    lines = z.split()
    for i in range(len(lines)):
        lines[i] = lines[i].strip(',')
    for pat in range(len(patient_list)):
        if patient_list[pat][0] == lines[1]:  # Checks if the patient is in the list.
            disease = patient_list[pat][2]  # Assigned the disease variable to use later.
            accuracy = patient_list[pat][1]  # Assigned the accuracy variable to use later.
            incidence = patient_list[pat][3]  # Assigned the incidence variable to use later.
            accuracy = round(float(accuracy.strip('%')), 4)  # "%" symbol is deleted from
accuracy, its value is made into float and rounded.
```

```
            a = round(1 - (accuracy / 100), 4)  # Accuracy is divided by 100, subtracted from 1
then rounded and assigned to "a".

            b = incidence.split('/')  # Incidence is split into two parts by the symbol "/" and
assigned to "b".

            b = float(b[0]) / float(b[1])  # First member of b is divided by the second member and
assigned to "b".

            sum = a + float(b)  # The sum of a and b are calculated and equalized to "sum".

            prob = float(b) / sum  # b is divided by sum to find the probability and we assign it to
"prob".

            if float(prob * 100) == int(prob*100):  # This part it for making the probability an
integer if the decimals are zero.

                prob = "{: .0%}".format(float(prob))

            else:

                prob = "{: .2%}".format(float(prob))  # Probability is transformed to percentage.

            global probability_number  # probability_number is made global to use in the
recommendation function.

            probability_number = prob.strip('%')  # We delete the "%" symbol from prob and
assign it to probability_number.

            global reco  # reco is made global to use in the recommendation function.

            if reco == 1:  # Breaks if reco is 1 to not write the probability when recommendation is
called.

                break

            else:

                save_output("Patient {} has a probability of{} of having {}.\n".format(lines[1],
prob, disease))

                break

    else:

        save_output("Probability for {} cannot be calculated due to absence.\n".format(lines[1]))




# This is the 'recommendation' function. When called, it compares patient's disease
probability and treatment risk and makes a suggestion.

def recommendation(r):

    row = r.split()

    for i in range(len(row)):

        row[i] = row[i].strip(',')
    counter = 0  # counter is used here for the same reason as in the patient_remove function.
    for pat in range(len(patient_list)):
        if patient_list[pat][0] == row[1]:
            risk = (patient_list[pat][5]).strip('%')
            global reco
            reco = 1  # reco is assigned to 1 to break the for cycle in the probability function.
            probability(r)  # probability function is called to calculate the disease probability.
            if float(probability_number) > float(risk):  # If probability is greater than risk, system
suggests the treatment. If not, it doesn't suggest.
                save_output("System suggests {} to have the treatment.\n".format(row[1]))
                reco = 0  # reco is assigned to 0 to neutralize the code.
            else:
                save_output("System suggests {} NOT to have the treatment.\n".format(row[1]))
                reco = 0  # reco is assigned to 0 to neutralize the code.
        else:
            counter = counter + 1
    if counter == len(patient_list):
        save_output("Recommendation for {} cannot be calculated due to
absence.\n".format(row[1]))


# This is the 'listing' function. When called, it makes a table of the patients' data in the list.
# It makes exceptions for words that are too long or too short.
def make_list():
    save_output("Patient\t""Diagnosis\t""Disease \t\t""Disease \t""Treatment\t\t""Treatment\n"
                "Name\t""Accuracy\t""Name\t\t\t""Incidence\t""Name\t\t\t""Risk\n"
                "-------------------------------------------------------------------------\n")
    for patient in patient_list:
        for i in range(len(patient)):
            patient[i] = patient[i].strip(',')
        if patient[4] == 'Surgery':
            save_output("{}\t{}\t\t{} \t{}\t{} \t\t{}\n".
                        format(patient[0], patient[1], patient[2], patient[3], patient[4], patient[5]))
        elif patient[2] == 'Lung Cancer':
            save_output("{}\t{}\t\t{} \t{}\t{}\t{}\n".
                        format(patient[0], patient[1], patient[2], patient[3], patient[4], patient[5]))
        elif patient[4] == 'Targeted Therapy':
            save_output("{}\t{}\t\t{}\t{}\t{}{}\n".
                        format(patient[0], patient[1], patient[2], patient[3], patient[4], patient[5]))
        elif patient[0] == 'Su':
            save_output("{}\t\t{}\t\t{}\t{}\t{}\t{}\n".
                        format(patient[0], patient[1], patient[2], patient[3], patient[4], patient[5]))
        else:
            save_output("{}\t{}\t\t{}\t{}\t{}\t{}\n".
                        format(patient[0], patient[1], patient[2], patient[3], patient[4], patient[5]))


# This is the 'reading inputs' function. It reads the commands and calls the required functions.
def read_input():
    with open(reading_file_path, 'r') as reading:
        for n in reading:
            if n.startswith("create "):
                patient_create(n)
            if n.startswith("remove "):
                patient_remove(n)
            if n.startswith("probability "):
                probability(n)
            if n.startswith("recommendation "):
                recommendation(n)
            if n.startswith("list"):
                make_list()


read_input()
```

4.2 Time Spent on This Assignment

Time spent on analysis	During the analysis of this assignment, I read the pdf file and all the attachments thoroughly and this took approximately 1 hour.
Time spent on design and implementation	After understanding the problem, designing and implementing a solution was the part that took the most time. 7-8 hours were approximately how much this section took.
Time spent on testing and reporting	Since the code was done, I wrote different scenarios to my input file and tested to see if my code was working properly. In this step, there were some errors I detected and fixing these with the whole testing part took around half an hour. The reporting part took about 8 hours.

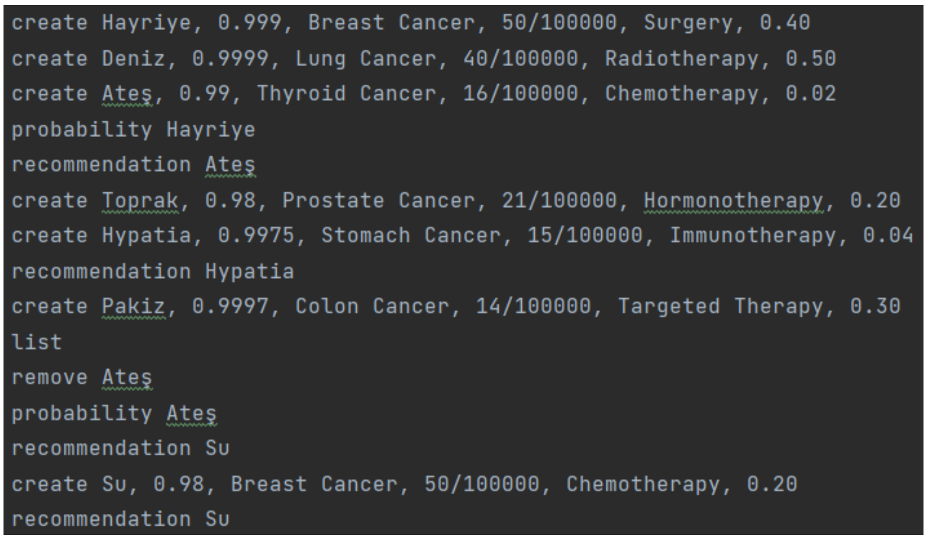
4.3 Reusability

While writing my code I tried to put relevant variable names and defined my functions with names that correspond with what they do when called. Therefore, I believe that my code is reusable.

5 User Catalogue

In this section, how to use the Doctor's Aid system is explained to users.

First you need to create an input text file called doctors_aid_inputs.txt and an output text file called doctors_aid_outputs.txt. You are going to be expected to write your commands to this input file. Here is an example on how the input file could look like:



```
create Hayriye, 0.999, Breast Cancer, 50/100000, Surgery, 0.40
create Deniz, 0.9999, Lung Cancer, 40/100000, Radiotherapy, 0.50
create Ateş, 0.99, Thyroid Cancer, 16/100000, Chemotherapy, 0.02
probability Hayriye
recommendation Ateş
create Toprak, 0.98, Prostate Cancer, 21/100000, Hormonotherapy, 0.20
create Hypatia, 0.9975, Stomach Cancer, 15/100000, Immunotherapy, 0.04
recommendation Hypatia
create Pakiz, 0.9997, Colon Cancer, 14/100000, Targeted Therapy, 0.30
list
remove Ateş
probability Ateş
recommendation Su
create Su, 0.98, Breast Cancer, 50/100000, Chemotherapy, 0.20
recommendation Su
```

There are five commands that you can give. These commands are create, remove, probability, recommendation and list.

Using the Create Command:

This command is for adding a new patient to the system. You must write the data in the order as follows: patient name, diagnosis accuracy, disease name, disease incidence, treatment name, treatment risk. These should be separated by commas. Diagnosis accuracy and treatment risk must be floating-point numbers. Disease incidence must be in a division format. Here is an example: "create Toprak, 0.98, Prostate Cancer, 21/100000, Hormonotherapy, 0.20". The code would write "Patient Toprak is recorded." to the output file. If we were to use the create command on Toprak again, the system would write "Patient Toprak cannot be recorded due to duplication.".

The disease name must consist of two words because, in the code the two words are combined into one element and with Targeted Therapy being the only exception, the treatment name must consist of only one word, otherwise errors will occur.

Using the Remove Command:

The remove command is for deleting an existing patient from the system. Simply writing "remove (patient name)" to the input file, will check if that patient is in the list and if so, delete the patient. "remove Toprak" command would write "Patient Toprak is removed." if Toprak were to be in the list. If not "Patient cannot be removed due to absence." would be written to the output file.

Using the Probability Command:

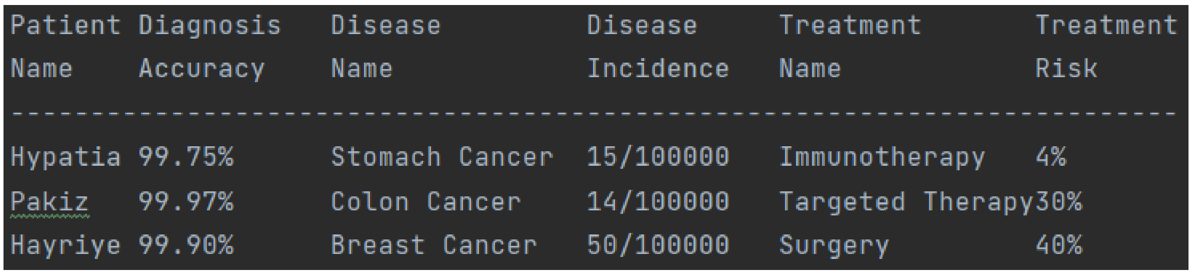
This command calculates and writes you the probability of the patient actually having the disease. By writing "probability (patient name)" you can see that patient's probability of having the disease if they are in the list. If not "Probability for (patient name) cannot be calculated due to absence." is written. "probability Toprak" would give us "Patient Toprak has a probability of 1.04% of having Prostate Cancer." if Toprak is in the list. If not "Probability for Toprak cannot be calculated due to absence." would be written to our output file.

Using the Recommendation Command:

The recommendation command is for giving a suggestion on whether the patient should take the suggested treatment or not. You may use this command by simply writing "recommendation (patient name)". "recommendation Toprak" command would give us "System suggests Toprak NOT to have the treatment." or "Recommendation for Toprak cannot be calculated due to absence." depending on Toprak's existence in the list. If the patient's disease probability is higher than the treatment risk, "System suggests (patient name) to have the treatment." is written to the output file.

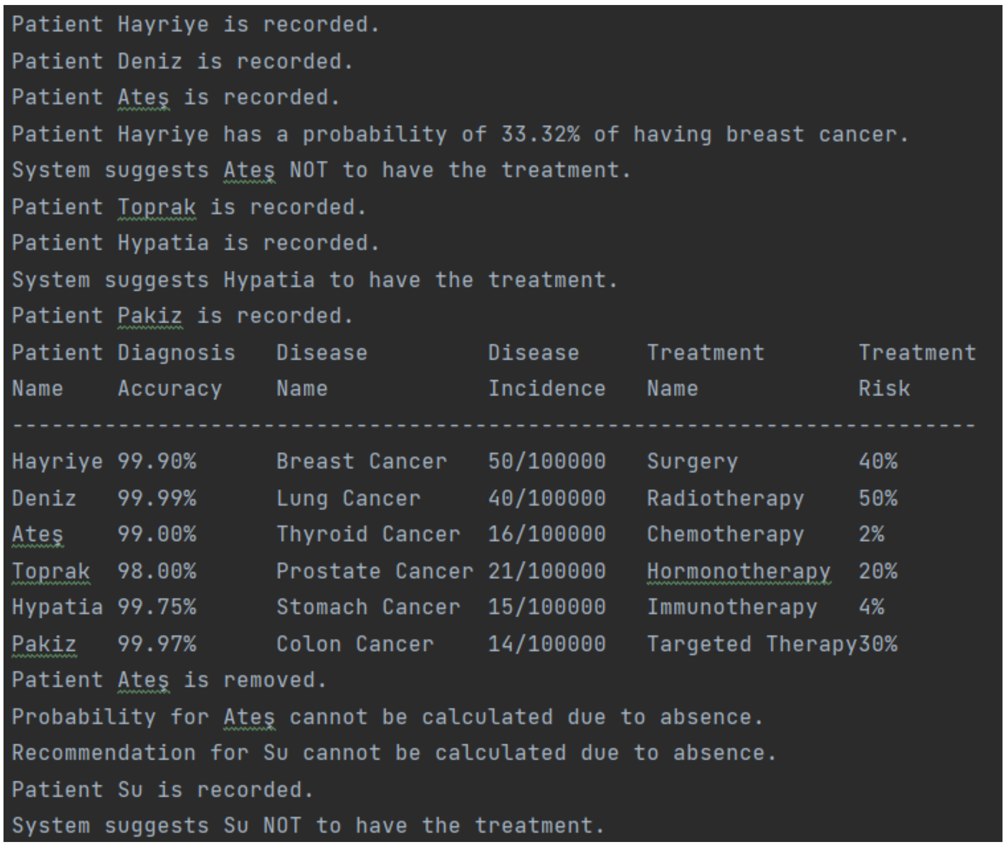
Using the List Command:

This command creates a table with the data in the system. You may use this command by typing "list" to the input file. Let's think of a scenario now. If you were to write "create Hypatia, 0.9975, Stomach Cancer, 15/100000, Immunotherapy, 0.04", "create Pakiz, 0.9997, Colon Cancer, 14/100000, Targeted Therapy, 0.30" and "create Hayriye, 0.999, Breast Cancer, 50/100000, Surgery, 0.40". and then write "list"; the list command would write this table:



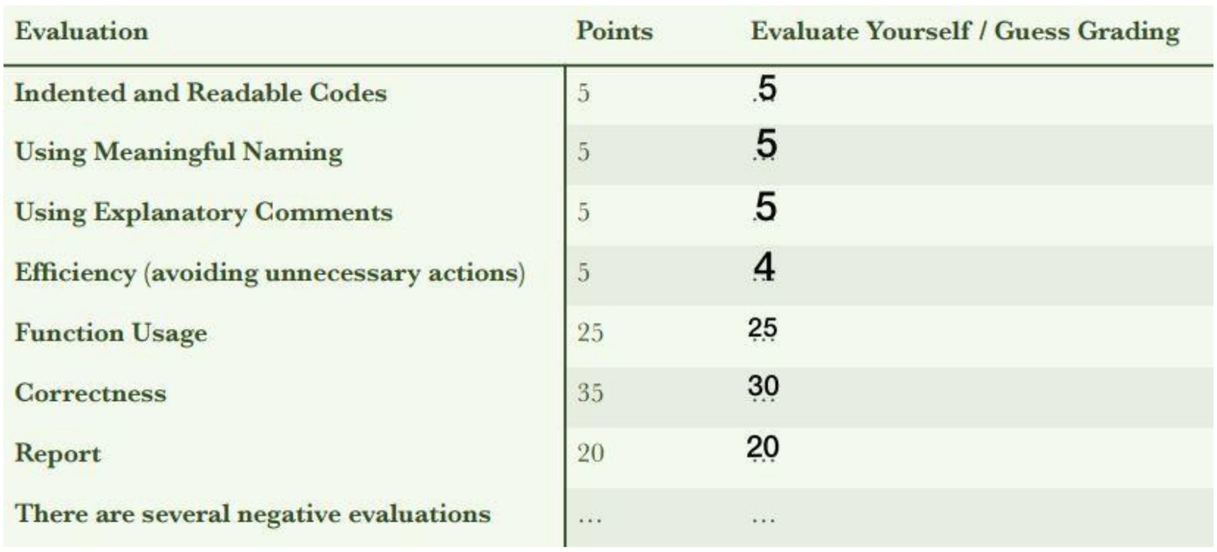
```
Patient Diagnosis   Disease         Disease     Treatment       Treatment
Name    Accuracy    Name            Incidence   Name            Risk
-------------------------------------------------------------------------
Hypatia 99.75%      Stomach Cancer  15/100000   Immunotherapy   4%
Pakiz   99.97%      Colon Cancer    14/100000   Targeted Therapy30%
Hayriye 99.90%      Breast Cancer   50/100000   Surgery         40%
```

Here is how the output file would look if you write the inputs given in the very beginning of the user catalogue:



```
Patient Hayriye is recorded.
Patient Deniz is recorded.
Patient Ateş is recorded.
Patient Hayriye has a probability of 33.32% of having breast cancer.
System suggests Ateş NOT to have the treatment.
Patient Toprak is recorded.
Patient Hypatia is recorded.
System suggests Hypatia to have the treatment.
Patient Pakiz is recorded.
Patient Diagnosis   Disease         Disease    Treatment       Treatment
Name    Accuracy    Name            Incidence  Name            Risk
-------------------------------------------------------------------------
Hayriye 99.90%      Breast Cancer   50/100000  Surgery         40%
Deniz   99.99%      Lung Cancer     40/100000  Radiotherapy    50%
Ateş    99.00%      Thyroid Cancer  16/100000  Chemotherapy    2%
Toprak  98.00%      Prostate Cancer 21/100000  Hormonotherapy  20%
Hypatia 99.75%      Stomach Cancer  15/100000  Immunotherapy   4%
Pakiz   99.97%      Colon Cancer    14/100000  Targeted Therapy30%
Patient Ateş is removed.
Probability for Ateş cannot be calculated due to absence.
Recommendation for Su cannot be calculated due to absence.
Patient Su is recorded.
System suggests Su NOT to have the treatment.
```

6 Grading Table



Evaluation	Points	Evaluate Yourself / Guess Grading
Indented and Readable Codes	5	5
Using Meaningful Naming	5	5
Using Explanatory Comments	5	5
Efficiency (avoiding unnecessary actions)	5	4
Function Usage	25	25
Correctness	35	30
Report	20	20
There are several negative evaluations	…	…